Document No.1550-001
30 April 1990

(2)

Repository Guidebook (Final) Technical Report for the Software Technology for Adaptable, Reliable Systems (STARS) Program

Contract No. F19628-88-D-0032

Task IR40: Repository Integration

CDRL Sequence No. 1550

R. H. Ekman
(Author)

30 April 1990

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

IBM Federal Sector Division
800 North Frederick Avenue
Gaithersburg, MD 20879

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 30, 1990	Final

4. TITLE AND SUBTITLE
Repository Guidebook (Final) Technical Report

5. FUNDING NUMBERS
C: F19628-88-D-0032

6. AUTHOR(S)
R. Ekman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
IBM Federal Sector Division
800 N. Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
CDRL Sequence No. 1550

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A guide to software reuse using the STARS Repository. This document contains the IBM STARS Repository Guidebook, STARS Repository User's Guide, and STARS Reusability Guidelines. Each is described below:

IBM STARS Repository Guidebook. A guide to the STARS Repository, providing high-level information for all users -- component reusers, component suppliers, and repository administrators. The Guidebook is organized according to the specific roles that users perform when using the system.

STARS Repository User's Guide. A guide on how to access and use the STARS Repository. It provides the basic information needed to use the repository software, but it is not a comprehensive guide to the VAX computer, on which the repository is built.

STARS Reusability Guidelines. A set of Ada coding guidelines for component development that emphasize reusability. Code that follows these guidelines will be easier to reuse on multiple project platforms. Many examples are provided illustrating the guidelines. (KR)

14. SUBJECT TERMS	15. NUMBER OF PAGES
STARS, software reuse, software reuse library, Ada coding guidelines, Ada	201
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Abstract

This technical report (CDRL Sequence No. 1550) addresses subtask IR40.1.1 (Guidelines, Procedures, and Standards) of the STARS Delivery Order Task IR40 (Repository Integration). It includes, as an attachments, the IBM STARS Repository Guidebook, the STARS Reusability Guidelines, and the IBM STARS Repository User's Guide. This report presents the methods used and lessons learned in developing the guides and the Repository. It is a revision of the draft Guidebook (CDRL Sequence No 1540).

The Repository guides define

- How to use the IBM STARS Repository,
- Resources and capabilities of the Repository,
- Component standards for admission to and promotion within the Repository,
- Procedures for submission and usage of repository components,
- Processes involved in managing repository assets, and
- The reuse process in the context of the STARS repository.

Preface

Due to its collective purpose and broad subject coverage, this report and the attached guides contain contributions from numerous individuals and relies on the work presented in many other documents and articles. Key among these are: Wendell Young and Gary Turner, authors of earlier IBM STARS documents; and all the developers in the IBM STARS IR40 team: Matt Bieri, Carol Heymann, Pamela Metcalf, Barbara Morck, Kyle Kennedy, Steve Kutoroff, Mike Puhlmann, and Tom Ward.

This report was originally scheduled to be completed on January 31, 1990, but due to the allocation of personnel to other STARS critical tasks and the extension of the STARS R-increment it was not completed until April, 1990. The attached guides will be updated at the end of the R-increment extension to reflect the final R-increment Repository.

This report was developed by the IBM Systems Integration Division, located at 800 North Frederick Ave., Gaithersburg, MD 20879. Questions or comments should be directed to the author, Robert W. Ekman at (301) 240-6431, or to the IBM STARS Program Office.

Table of Contents

Introduction 1
Identification 1
Purpose 1
Terminology 1
Task Description 3
Information Sources 3

Guidebook Concept 5
Organization/Structure 5
Content 6
Assumptions 7

Conclusions 8
Lessons Learned 8
Repository Lessons 8
Reusable Component Lessons 11
Software Development Life-Cycle Lessons 12
Recommendations 15

References 16

Bibliography 17

Acronyms 21

Attachments 22

Introduction

Identification

This technical report, IBM Contract Data Requirements List (CDRL) Sequence Number 1550, addresses the Software Technology for Adaptable, Reliable Systems (STARS) Subtask IR40.1: Functionality, as defined in the *STARS Prime Contractor, Delivery Order 2* [DO2], Task IR40: Repository Integration, and satisfies, in part, the *IBM STARS Program Administration Plan* [IBM1280], Subtask IR40.1.1: Guidelines, Procedures, and Standards.

This report is a revision of the *Repository Guidebook (Draft)* [IBM1540], which was delivered under the same subtasks. Refer to the Program Administration Plan for a complete list of deliveries under the IR40 task and its subtasks.

This report follows the format requirements of MIL-STD-847B, as specified in the Technical Report Data Item Description (DID) DI-S-3591/A.

Purpose

This report was prepared to satisfy the delivery requirements of the STARS IR40 Repository Integration task and to provide a place to record the experiences gained from the task. The Guidebook, Guidelines, and User's Guide are attached as a separate stand-alone documents that can be distributed to users of the Repository.

The guides are a collection of descriptions, procedures, and guidelines that covers all views of the IBM STARS Repository, including system access, user roles, repository capabilities, component contributions, component coding, and repository administration. The guidelines and procedures have been applied to the IBM STARS team tasks during the STARS prime contract R-increment and will be applied to IBM STARS tasks in subsequent increments.

Terminology

The following is a list of terms and definitions used in this report. An appreciation of these terms will help in understanding the IR40 Repository Integration task. We have chosen to present them in a descriptive order which supports the relationships between the terms and permits serial reading.

component — A collection of related work products to be used as a consistent set of information. Software work products can include specifications, design, source code, machine code, reports, compilation units, code fragments and other components. A component is the focus of a repository and is intended to be reused. It is the primary type of object found in a repository. Also referred to as an **asset**, or **resource**.

part — An element of a component, such as design documents, source code, test information, and data rights. While a part may be copied or browsed, when stored in a repository it is always associated with a component.

reuse	The application of existing solutions, as captured in components, to a problem other than the one for which they were originally built.
repository	An element of the software engineering environment in which software work products and information about them are stored. Its primary purpose is to support the reuse of components through a process of component submission and extraction. Also referred to as a **reuse library**.
repository system	An instance of the software engineering environment, including computer hardware, operating systems, software tools, standards, and procedures that together provide a complete repository capability. These capabilities include acquiring, storing, managing, retrieving, and dispensing software work products and information about them to potential reusers. A repository system may contain several repositories, each dealing with a different problem domain.
depository	A repository or part of a repository whose contents are deposited as is, with few or no constraints. A depository is primarily a storage facility. Quality and usability are unpredictable; the burden is on the user to find and evaluate useful items.
organized repository	A repository whose contents are documented and organized in a comprehensive manner as components and parts. Finding, reviewing, and extracting components are supported. The quality and usability remain unpredictable.
filtered repository	A repository whose components are subjected to standards on form, content, quality, and consistency. This is not a physical partition from the organized repository, rather it is a documented higher level of confidence in an existing organized component.
certified repository	A repository whose components exhibit the highest level of confidence. A certified component permits the proving of correctness in a solution that reuses the component. Aspects of security, ownership, distribution, and user set take on greater importance.
reuser	A person who reviews the contents of a repository and extracts components for reuse. The common user of a repository. Also referred to as a **user**.
supplier	A person who places components into a repository. Also referred to as a **contributor**, or **submitter**.
librarian	A person who coordinates, organizes, and controls the contents of a repository. Also referred to as a **system administrator**, **database administrator**, or **repository administrator**. This person frequently provides a first level help for other users.
topic specialist	A person who is responsible for the evaluation of components and their promotion to the filtered or certified status. This person is an expert in the repository domain and has reviewed many of the components in the repository. Also referred to as a **domain expert**, or **technical consultant**. This person frequently provides a second level help for other users, via reference from the librarian.
filtering	The act of evaluating components and promoting them to filtered status.
gatekeeper	An automated capability that facilitates filtering.

These definitions are derived from definitions in several STARS Prime contract documents [IBM380, IBM460, Boeing330, Unisys340], and from several industry publications [IEEE729, SPCStyle]. A complete list of the IBM STARS Repository terminology is given in the Guidebook Glossary.

Using these terms, the IBM STARS Repository is defined as a repository system, that supports the reuse of components in the domain of software engineering environments. The Repository contains a full suite of capabilities and guidelines that are required by users of the system. The Repository supports multiple levels of component confidence (depository, organized, filtered, and certified) and has facilities to promote components through these levels.

Task Description

This report and attached guides are the results of efforts by members of the IBM STARS IR40 task team over the eight month performance period of the task. We conducted numerous reviews, meetings, and discussions on the material presented. We iteratively developed and evaluated operating prototypes to experiment with ideas and validate approaches.

Several existing repositories were reviewed and exercised, including the Ada Source Repository (ASR), the AdaNET information service, the Boeing STARS Repository, and the Army RAPID component repository. These systems provided insight into performance, content, and user issues.

Besides the producing these guides, our activities have

- Refined the state-of-the-art in repository and reuse technology,
- Improved our understanding of operational repository systems,
- Made our experiences available to government and industry through our delivery items, and
- Established an operational repository to support future IBM STARS tasks.

Information Sources

The primary sources for this report and the attached guides were the IBM STARS documents: CDRL 380 *Consolidated Reusability Guidelines* [IBM380], CDRL 460 *Repository Guidelines and Standards* [IBM460], and CDRL 1540 *Repository Guidebook (Draft)* [IBM1540]. These documents were derived from several older documents, some of which were reviewed again during this task. In addition, this report, the guides, and the Repository development were influenced by numerous articles and papers on software reuse and repository technology.

The following is a structured list of the information sources that were used during this task. Complete references for the documents are in the Bibliography for this report. Copies of all the STARS documents can be found in the IBM STARS Repository.

1. STARS Q-Increment Documents
    a. IBM
        1) Task Q12
            a) CDRL 460: *Repository Guidelines*
            b) also, but of lesser importance:
                i. CDRL 470: *Repository System Plan*
                ii. CDRL 480: *Demonstration Script*
                iii. CDRL 490: *Demonstration Results*
                iv. CDRL 500: *Repository Specifications*
                v. CDRL 510: *Repository Implementation Report*
                vi. CDRL 520: *Repository Configuration Control Plan*
                vii. CDRL 530: *Prototype Repository Implementation*
        2) Task Q3
            a) CDRL 110: *Environment Capabilities* (included CDRL 100: *Matrix*, and CDRL 90: *Requirements*)
        3) Task Q9
            a) CDRL 380: *Reusability Report* (included CDRL 370: *Analysis*, and CDRL 360: *Guidelines*)
    b. Boeing
        1) Task Q12
            a) CDRL 320: *Repository Guidelines*

b) also, but of lesser importance:
   i. CDRL 330: *Initial Capabilities*
   ii. CDRL 380: *Configuration Control*
   iii. CDRL 400: *Repository Report*
   iv. CDRL 420: *Demonstration Plan*
c) peer review of:
   i. IBM Task Q3 CDRL 100
   ii. IBM Task Q9 CDRL 380

c. Unisys
   1) Task Q9
      a) CDRL 340: *Reusability Guidelines*

2. STARS R-Increment Documents

   a. Task IR40: *Repository Integration Proposal*
   b. Task IR40 CDRL 1540: *Repository Guidebook (draft)*
   c. Task IR40 CDRL 1580: *Taxonomy Report*
   d. Task IR40 CDRL 1590: *Repository Prototype Specifications*
   e. Task IR40 CDRL 1600: *Repository Version Description Document*
   f. Task IR40 CDRL 1610: *Repository Demonstration*
   g. Task IR10 CDRL 1440: *Repository Operations*
   h. Task IR10 *Repository User's Guide*
   i. Task IR11 CDRL 1460: *Repository Policy and Procedures*
   j. Task IR11 CDRL 1470: *Repository Operations and Procedures*

3. Other IBM Documents

   a. IBM Systems Integration Division Owego *Reusability Guidelines*
   b. IBM Systems Integration Division work in reuse and repositories
   c. IBM Corporate work in reuse and repositories

4. Other Key Documents

   a. Software Productivity Consortium (SPC) *Ada Style Guidelines*
   b. Naval Research Laboratory (NRL) *Reusability Guidelines*
   c. SoftTech *Reusability and Portability Guidelines*

5. Tools/Product/System Documentation

   a. VAX/VMS
   b. AdaMAT
   c. Oracle
   d. ASR
   e. RAPID
   f. AdaNET

6. Miscellaneous

   a. IBM STARS Program Office notes
   b. Developer's notes
   c. Feedback from reviewers
   d. Operational repository procedures

Guidebook Concept

The goal for the Guidebook is to define the reuse process and gather in one place all the references, standards, guidelines, and procedures related to the operation and use of the IBM STARS Repository. The Guidebook and related guides will be handed out to software engineers who want an introduction to the Repository and will be using the system. The guides will be stored in electronic form on the Repository system.

Organization/Structure

The Guidebook was prepared in separable parts to assure accessibility and convenient use. The arrangement of its constituent parts is a matter of practicality, dictated by the source of the information.

The Guidebook has become a suite of documents, with the Guidebook at the center. This technical report is considered a 'preface' to the Guidebook and not required by any Repository user. The Reusability Guidelines were separated from the Guidebook to provide guidance for component developers. The User's Guide was also made separate from the Guidebook to permit changes in operational aspects of the Repository, without republishing the complete guidebook.

The Guidebook and its related guides are living documents. Each part has a potential for change based on its relationship to the Repository system and reuse as practiced by the IBM STARS team. Updates will be prepared as these changes dictate.

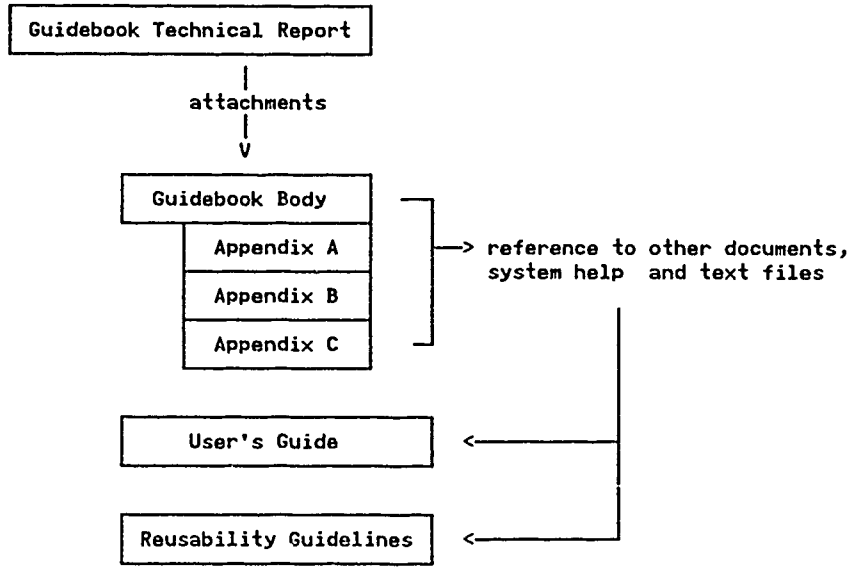

Figure 1. Graphic Model of the Guidebook Parts

In organizing the body of the Guidebook, we considered several dimensions or views of the Repository and the reuse practices. These included:

- the roles of system users (reuser, supplier, administration)
- repository levels (organized, filtered, certified)
- requirements for entry and promotion
- procedures to be followed
- system design and architecture

From discussions of these alternatives, we decided that the *roles* dimension was the best way to present the material, and that roles would be the focus of the body of the Guidebook. The other views tended to have long lists of detail and did not help the user in organizing their approach to reuse. The other views are presented in the appendices or by references to other material.

Content

The Repository Guidebook defines

- Resources and capabilities of the STARS Repository,
- Procedures for submission and usage of repository components,
- Standards for promotion of components within the Repository,
- The processes involved in managing repository assets, including component configuration management, problem tracking, promotion, and entry and exit criteria, and
- The reuse process in the context of the STARS repository, covering the admission, adaptation, integration and testing of reusable components, as well as methods for locating and selecting reusable components.

While producing the Guidebook, reviewers asked for many topics to be included. We realized that attempting to cover all of them would be difficult, but we also felt that most of the request were real needs for some part of the user population. The challenge was to organize the material and structure the documents to give an meaningful impression of the repository, while preserving enough detail to be helpful for a particular user with a specific problem.

The following is the list of topics we addressed in this task and where we decided to place it.

Guidebook material	Location
people roles and activities	body
glossary	body
definitions	body, report
references	body, report
acronyms	body, report
relationship to STARS tasks	report
guidebook concept	report
assumptions	report
lessons learned	report
bibliography	report
summary of resources and services	appendix
gate definitions (filter list)	appendix
user directions	appendix, attachment
coding guidelines for reusability	appendix, attachment
operational policies of our system	appendix, reference
data requirements and database tables	reference
help information	reference
system screen images	reference
how to do reuse	reference
where reuse fits in lifecycle	reference
configuration management	reference
access security	reference

Assumptions

The following assumptions pertain to the guidelines and procedures presented in the Guidebook.

- The Guidebook and associated guides support reuse within the context of a software intensive project. They may be used to establish a starter set of guidelines, but does not form a complete set. Your project should modify and augment these guidelines to address domain and system specific issues.
- Issues of security, proprietary rights, and component ownership are beyond the scope of this version of the Guidebook, but may be found in referenced documents.
- The coding guidelines are directed primarily toward Ada software engineering. While some guidance may apply to a broader class of components, the focus is on Ada software. The guidelines can thus treat specific aspects of Ada that relate to reusability.
- No assumptions are made about design methodology and relationships to reuse.
- Reuse is assumed to include tailoring of reusable components. It is likely that a significant amount of reuse will involve some tailoring of the component code. Thus, there are guidelines to promote tailorability. Note that *tailoring* should be distinguished from the broader term *maintenance*, which not only includes tailoring but also includes the idea of corrective maintenance (fixing bugs).

Conclusions

The IBM STARS IR40 task team has worked for nine months developing the prototype repository system that is documented by the Guidebook. The team has included up to ten engineers, in three different locations, developing the system on several different platforms. The team developed four releases of the system following an iterative refinement prototyping method. The experience gained during this task is the basis for the conclusions in this report.

Lessons Learned

Several situations arose during the writing of the Guidebook and the development of the Repository which we believe contain "lessons learned" that could be applied by the greater software development community. We have collected these lessons and classified them according to software engineering technology areas. In many cases, the lessons were previously stated in draft documents and status reports. Our purpose in this report is to gather all the IR40 team lessons learned into one place.

The lessons are stated as a recognized problem or situation, followed by our observations on the problem, an implemented solution, or alternative potential solutions.

Repository Lessons

At a high level, developing a repository has issues and design similar to the development of any on-line application with a large database. But when you consider a repository system as a special element in a software engineering environment, unique problems appear.

From an object-oriented view, the life-cycle of objects in a repository is similar to the life-cycle of objects in a software engineering configuration management system. The differences are that the repository objects are less frequently updated, that there are weaker relationships between the objects, and that ownership is harder to establish. A software engineering environment deals with objects as a project library, while a repository deals with objects as a public library.

- *It is hard to describe the Repository due to the varying experiences of the users and the different roles they play in the reuse paradigm.*

  The critical information in a repository can be viewed from many different perspectives. For example, the coding guidelines can be viewed from the perspective of the developer of reusable components, the submitter who presents the component for admission to the Repository, the topic specialist who analyzes the component, and the reuser who extracts the component from the Repository. The developer follows the guidelines in creating the initial work product. The submitter and topic specialist reference the coding guidelines to confirm that they have been adequately followed. The reuser will want to know what coding guidelines are preferred and supported by the repository to facilitate the usability analysis.

  To address this situation, we organized the body of the Guidebook based on user roles. We also organized the Repository capabilities based on user roles. This organization is apparent in the selection menus and the system design.

- *There were few aids to make it easy to follow guidelines and there were no aids that support different user roles.*

Each role is defined with a set of procedures which generally refer to one or more sets of guidelines. The guidelines are often too extensive or too broad for regular reference. We found that checklists can be used to promote consistency and completeness and to customize guidelines and procedures to a particular environment or application. The checklists made it possible to provide a single set of general guidelines and use the more succinct checklist to place emphasis on the important guidance. For example, the coding guidelines are customized for the component developer by the coding guidelines checklist. Further customizing could be achieved by providing portability and reusability checklists which emphasize those coding guidelines that topic specialists should consider in evaluating a component for admission to the repository.

- *The relationships between Guidebook information, its sources, and users are complex.*

  The intellectual, physical, and temporal relationships among the Guidebook information, its sources, and its users are quite complex, making it difficult to provide the information in a meaningful and manageable manner. It seems clear at this point, that the CDRL delivery mechanism is not the complete answer, since the information in deliveries, as a whole, changes continuously. The fundamental problem is the paradox that specific information is both more useful to Repository users and more volatile than more general information. For example, the names of individuals identified as points of contact or assigned to other roles are more likely to change than the role abstraction.

  The potential users of the information contained in this Guidebook are many and diverse. For this information to be used, it is necessary to achieve several goals. It must be provided accurately, when it is needed, where it is needed, in a form that is meaningful to the user of the information, and in a manner that is as unobtrusive as possible. Indications (such as time-stamp or version number) should also be provided to give the user confidence that the information is as timely and accurate as it needs to be. This implies that information updates must be rapidly disseminated to end users. Ways must be found to improve the packaging and delivery of the Guidebook and its information content.

- *It was difficult to synchronize updates to information, even when a direct relationship was understood.*

  Information sources also influence the packaging objectives for the information since it is desirable (though not necessary) for information provided by a given source to be (logically) co-located and for updates to occur as atomic operations to ensure the continuing validity of the complete document.

  This becomes increasingly important as various manual functions are automated. We have used information modelling techniques to develop our understanding of the relationships and predict the synchronization problems.

- *The development guidelines, on-line definition of the guidelines, and the automation of component review regarding the guidelines were difficult to coordinate.*

  In establishing a complete repository, we developed guidelines for components and procedures for the component life-cycle within the Repository. These guidelines and procedures appeared in several places, such as the Guidebook, on-line help, and rules for Ada code metric analysis. In addition, they were refined and modified over time.

  This situation made it difficult to coordinate the guidelines development. At times the guidance in one place contradicted the guidance in another place. By the end of the R-increment, we believe most of the irregularities have been addressed, but we have not established an easy way to keep the guidance coordinated and get a reasonable level of consensus.

- *An undefined glossary made documentation, discussions, and component evaluation difficult.*

  In the Repository, we needed to distinguish terms from their more general English definitions. We constructed a structure of terms and classifications within those terms, but we had difficulty in communicating our meanings.

We found many competing definitions for the same words. Similar type work in reuse and repositories is being conducted by many organizations and they have published their glossaries. These definitions overloaded our definitions and caused misunderstanding. Further, we contributed to the confusion due to our own refinement and elaboration of our definitions over time. These problems are not unusual in leading-edge efforts, but they are areas for concern and require attention.

Our answer to the terminology problems was to include our definitions in several key documents and reports. We reviewed some of the competing glossary lists and adopted their definitions where appropriate. But in the end, we were left with a set of definitions that work for us, but are neither "fish nor fowl" in any official world.

A similar issue appeared when reviewing documentation for software components. For example, it was difficult to understand and evaluate the "Dates" software package without a good understanding of horological terms, such as Julian date.

To answer the component understanding problem, we have suggested that components contain a glossary of keywords in all related documentation. There is a direct relationship between the documentation of the component glossary and the readability of the declarative section of the code.

- *The linear nature of hard-copy documents does not translate to on-line review very easily.*

  Books, manuals, and other hard-copy documents are inherently linear in physical organization. This results in difficult choices: Is it better to repeat textual information at the risk of annoying the reader who encounters repeated text, or is it better to provide references among the singular instance of a unit of text much like is done in an encyclopedia, putting the burden of locating the information on the reader?

  In the Guidebook, we chose the reference approach. This choice also reduced the maintenance and new material problems, much like subunits in a programming language.

  Some alternatives to the linear organization are network links between material as implemented in hypertext systems, and hierarchy tree models as implemented in many on-line help systems. These alternatives are desirable organizations for the Repository information system, but were displaced by more important tasks.

- *Automated component evaluation was only partially successful.*

  We attempted to construct an automatic filter for components going into the Repository and for evaluation of components within the the Repository. We found it time consuming and difficult to construct these automatic filters, and we found that they ultimately would only do part of the job.

  To evaluate the compilation status of a component, the bulk of the effort was consumed by review of the compilation order information, location of external packages and libraries, and similar type tasks. Running the compiler was trivial, and automation of just the compilation step did not save much effort.

  The review of documentation was all manual. We were not able to locate or build any tool to help with documentation evaluation. Because of the variety of documentation standards and styles, it is unlikely this area will be automated soon. There are some tasks, such as grammar and spelling checks, that can be supported by tools, but they do not help with the evaluation of documentation completeness.

  Review of Ada code was partially automated with the help of the tool AdaMAT, by Dynamics Research Corporation (DRC). This tool gave us counts of constructs that are considered "good" against a count of where the construct could have been used. The AdaMAT reports helped to suggest places and issues to review, but could not substitute for human inspection of the code.

Reusable Component Lessons

We populated the IBM STARS Repository with hundreds of components and thousands of related parts. This population task was usually accomplished by system administrative engineers, and occasionally by the component authors.

Some of the following lessons may seem obvious and the answers may appear to be the application of rigor and management techniques. But this is the area where we had the least success. We originally planned to have many more components and parts in the final repository. Every time we sat down to really build up the repository content, other things got in the way. The database structure kept changing, the submit mechanism was too hard to use, or the task was just too boring and labor intensive.

The following lessons present major issues from our evaluation of the problems in repository content management.

- *It is difficult to successfully and completely contribute a component from a third party.*

  The process of organizing the components in a repository is more difficult when the components are extracted from a source, rather than submitted by a supplier. For example, some of the components from the Ada Source Repository (ASR) have been organized on the IBM STARS repository. These components took much longer to organize than the components submitted by the STARS prime contractors, because the ASR librarian was not involved in meeting the IBM STARS repository entry requirements. On the other hand, some of the STARS prime components were well known by the people doing the entry to repository, and the time required to organize them was significantly less.

  This observation led us to put a greater emphasis on having the component author do the submission to the repository. The person who knows the most about a component is its author, and the repository submit process should take advantage of this knowledge. The submit process must be easy to use, capturing a large amount of information, which would be harder to acquire later on. A clear but not extensive set of guidelines and procedures must be established, and they must be supported by the system capabilities, especially at the component entry time.

  This observation also leads us to question the value of third party repositories that are currently available, such as the ASR, AdaNET, and the Ada Information Clearinghouse. These systems freely pass along components and information, and they support general communications, which is much needed in the Ada technology community. But without the explicit involvement of the authors in the cataloging and depositing of the components, the confidence is lowered in the ownership, completeness, and currency of the component. And confidence in the extracted component is significant in the success of reuse.

- *There is little incentive for engineers to contribute their work products to a repository outside their current project.*

  This lesson is not new. We have just recognized it again, and we want to reiterate the problem. A good reuse repository must contain some incentives that make people want to supply components for reuse. Further, our observations revealed that components that do not have the force of an incentivized supplier behind them, are probably not good candidates for a reuse repository.

  While not being terribly successful, we have attempted to place a repository submission subtask in all major tasks. The concept is that one of the measurements in completing a task is the submission of reusable components out of the task's workproducts into the repository. This is the model that we will follow in future IBM STARS tasks.

  A major disincentive to engineers in our repository model is the effort required to contribute a component. It takes as much as four hours to contribute a moderate size component. This seems excessive, but we believe it is justified. The effort to supply components is inversely proportional to the effort to reuse, and a primary objective of a repository system is to reduce the effort to reuse. A balance is required between the two efforts. A factor in this balance is

the realization that supply is a one-time effort and reuse is a recurring effort. Each reuse increases you return on supply cost investment.

- *The names of components and the names within components confused the repository administration and reusers.*

  Contributors frequently used names that are only meaningful to their environments. The names may be imbedded in their contributions and don't "flow through" to the repository system, or to the reuser's development system.

  This problem has became more apparent during final reviews of the Repository code and the Repository content. It can be a serious inhibitor to software reuse.

  We have established name guidance for the Repository development to reduce this problem. The initial guidelines are in the Guidebook. We anticipate refinement of these guidelines and additional guidelines as we learn more about the problem.

- *Software components without compilation order information are almost useless for the reuser.*

  This lesson may seem obvious, but some contributions to our repository (and many contributions to other repositories) have not included compilation order information. The submitter assumes the component is simple enough, such that anyone could compile it, or they rely on the naming convention to indicate compilation order. In some cases, it may be acceptable, but the absence of the compilation order information almost always causes confusion.

  Our guidelines require compilation order information and it is part of the evaluation process followed by the topic specialists. We do not bar contributions without compilation order information, but the components without it will unlikely make it to the filtered state.

Software Development Life-Cycle Lessons

During the development of the IR40 prototype repository system, we followed a rapid iterative refinement methodology. We did not follow the conventional waterfall life-cycle, rather we followed the emerging software-first life cycle. We eliminated almost all "standard" documents and took many shortcuts in order to produce the greatest amount of operating prototype capability in the shortest time. We wanted to produce the "look and feel" of a repository, based on a general environment architecture, with standard interfaces, all running on real data in a significantly populated database.

We believe the process we followed was acceptable and produced a cost effective repository system. We plan to continue to experiment with this process in future IBM STARS tasks.

We anticipated most of the problems we encountered, since we recognized our deviation from the norm, but we were surprised at how much effort and time was required to address them. There was always a strong feeling to stay with whatever we had, in both software and methods. There was a continuous and uneasy balance between resources available and planned tasks.

- *Operational, development, and test environments interfered with each other.*

  The Repository not only was developed in several locations and on several platforms, but also contained several parts which did not come together until we build an operational system, such as the database, menu system, and search mechanism. This is not unique to this project, but it did cause stress on the development cycle.

  We had to develop multiple databases, and populate them several times. Separate test environments needed to be maintained to allow testing to proceed without disruption of the existing repository data base.

  We developed several layers of the architecture simultaneously. To accomplish this, we copied significant amounts of source code into several development environments. Each environment had naming restrictions which forced us to use different naming patterns for the same source elements. These duplicate systems created a versioning maintenance problem. This also

lowered our confidence with the final product, because we could not guarantee that the migrated elements were unmodified.

- *Too few people tried the system too late for changes.*

  The development model we followed called for frequent execution of the evolving system by a wide user population. We accomplished this through full system releases to the STARS community and partial prototypes for a narrow set of users. We either demonstrated the prototype or asked the user to experiment with it by themselves. We captured remarks from demonstrations and problem reports from individual users, as our feedback mechanism.

  The problem we had was in striking a balance between preparing clean prototypes for a large set of users and rapid prototypes for a controlled group. Even though we maintained operational parts of the system throughout the task period, it still took significant efforts to put together fully operational systems. Complete systems were needed for the user to experience and appreciate the set of capabilities. In some cases, we put together partial systems which demonstrated particular ideas. These prototypes proved useful, but only to a small set of intimately involved users.

  We did not uncover any simple way to approach this balancing effort. Just continual attention to the basics of rapid prototyping is all we can offer as a lesson. Rapid iterative development is more successful if early prototype capabilities can be made available to a wide set of experienced users who have access to immediate and easy to use feedback mechanisms.

- *The Repository user interfaces were not consistent among embedded tools.*

  Rapid prototyping and integration frequently leads to systems with "bumpy" user interfaces. In constructing a rapid prototype, you must reuse components that were written for widely different situations. The user interfaces that accompany these components are usually inconsistent. They were written to take advantage of the perceived users experience and operational system. If, when searching for reusable components, you stay only within a single domain and platform, you will reduce this problem, but you will not be able to take advantage of other reuseable components. The development cycle becomes a trade-off between capability, usability, and time.

  During the development of the Repository, we concentrated on the balance between capability and usability. We attempted to extend the system capability with less regard for usability. This approach was helpful in developing the overall architecture and design, but when it came time to demonstrate the system, we discovered usability problems. The problems with usability caused us to lose "user hours" and reduced the feedback on the prototype, and thus reducing the benefit of this life-cycle.

- *Parts of the Repository system did not have clear ownership.*

  The final operational Repository is not a single unit, but rather a collection of executables, held together by a menu selection mechanism, relying on a database for much of the display window content. Included in the Repository are the Guidebook, guidelines, and numerous help screens. A significant amount of material was reused from the STARS Q-increment and incorporated into the Repository system. Furthermore, over the repository development period, there was considerable turnover in the IR40 team. These facts lead to situations in which there was uncertain ownership of elements of the Repository system.

  While the ownership problems were cleared up by the end of the R-increment, it was a noteworthy issue during development. Without clear ownership, rapid development can get out-of-hand. Effective configuration management methods should be used to get the most out of iterative development. When you reuse material that does not have clear ownership, you must be willing to take on ownership. You need to be concerned with ownership of guidelines and procedures to the same level as ownership of software.

- *Quality assurance was not consistent or occurred too late.*

  Even in a rapid prototype development cycle, as performed by the IR40 team, an independent quality assurance audit needs to be conducted. The audits conducted on the Repository

system, brought forth several potential problems that were taken care of before reaching the users. But the development cycle used on the Repository did affect the quality assurance tasks.

The timeliness of the quality assurance tasks was a problem. Frequently, we were well into the next iteration before the assurance tasks were complete. The system had matured and obviated many of the reported problems.

The second problem we had was consistency across the multiple sites and development platforms. Each group had their own way of handling quality assurance. The pressure to develop and demonstrate quickly, as called for in a rapid development cycle, dictated that we skip a more complete and orderly approach.

- *User initiated demonstrations were not effective.*

Two methods were used to get reviews of the Repository prototypes: general broadcasts to the user population that a feature or release was available, and personal face-to-face demonstrations by the developers. The mechanism of feedback for the general release was a problem reporting system and electronic mail notes back to the developers. The mechanism of feedback for the personal demonstrations were comments during the presentations and observations of the users actions. While feedback from both methods was helpful, the personal demonstrations had faster feedback and gave a better sense of the value of the feature being demonstrated.

Users seldom took time out of their busy schedule to exercise a prototype or new release under their own initiative. Unless a capability was absolutely needed to complete their tasks, we received very little response from the user community. Even when the effort was planned as peer review in their primary task, the self initiated exercise of a prototype was too late to affect development. The problem reporting and electronic mail feedback did provide a documented trail of problems and a measurable item. But the nature of the reports and notes were too detailed for prototype evaluation, and focused on errors and screen content. Serious problems and alternative solutions generally did not appear in this method of feedback.

Face-to-face demonstrations were conducted in offices between developers and users, in meetings with management and architects, in classroom environments, and at conferences to a wide audience. Very little specific documented feedback resulted from these demonstrations, but user impressions were important in evaluating the success of the prototype.

- *Communications were difficult in the IR40 development team.*

Part of the design and plan of the IR40 task was to experiment with distributed development among locations, development environments, and system architecture. But because of this distributed organization and structure, inadequate communications between the team members was a problem. Tom Peters [Peters82] and Fred Brooks [Brooks75] see communications as the most critical element in the success and excellence of a project. Our experience in the IR40 task agrees with their assertions.

We were not able to eliminate communications problems, but we did address the problem. Our approaches to reduce communication problems were to

- Use travel to the degree permitted in the contract,
- Take advantage of other travel plans to coordinate our work,
- Use conference calls, and
- Use electronic mail, especially over phone lines via the repository system.

A key element in a software development effort is the configuration management system. It is a common notion that these systems help with project communications. Unfortunately, because of the heterogeneous nature of the IBM STARS team network, configuration management systems actually hurt sharing information and communication among the team members. Each group within the team used locally available configuration management methods. The manual import/export capabilities of these systems made it difficult for team members outside the local group to review development.

Recommendations

In summary, the "lessons learned" can be stated in positive action statements as follows:

1. Repository Lessons

    a. Describe the Repository from user views.
    b. Establish guidance using support tools based on user views.
    c. Define relationships between user guidance information.
    d. Synchronize updates to user information.
    e. Coordinate guidance with automated tools.
    f. Publicize a glossary of Repository terms; supply a glossary with components.
    g. Use reference pointers to information sources.
    h. Plan for manual component evaluation, even with automated tools.

2. Reusable Component Lessons

    a. Get the component author or owner involved in the submit process.
    b. Seek an incentive for engineers to contribute their work products.
    c. Pay special attention to component names and embedded names.
    d. Supply compilation order information with components.

3. Software Development Life-Cycle Lessons

    a. Clearly separate operational, development, and test environments.
    b. Involved a large number of reviewers early in the development cycle.
    c. Strive for consistent user interfaces.
    d. Establish clear ownership of developed workproducts.
    e. Apply quality assurance consistently, even in rapid iterative development.
    f. Use controlled small group demonstrations.
    g. Spend significant effort on communications within the development team.

In addition, the development of the Repository and associated Guidebook has lead to the following general recommendations.

- Spend more time with the users, or your user interface will be uninviting.
- Spend effort smoothing the contribution mechanism or you will not have anything to reuse.
- Put aside significant time and funds to populate the repository and evaluate components. Our experience suggests about:
    - 1hr per simple component (component with no dependency on other components) for submission to the repository.
    - 4hrs per simple component for evaluation and filtering.
    - 20hrs per simple component for certification.

References

The following is a list of references in this technical report in the order that they are cited. The attached guides contain separate lists.

[IBM370] IBM Systems Integration Division, *Reusable Component Data Analysis*, CDRL Sequence No. 0370, February 10, 1989.

[IBM380] IBM Systems Integration Division, *Consolidated Reusability Guidelines*, CDRL Sequence No. 0380, March 21, 1989.

[DO2] United States Department of Defense, Electronic Systems Division USAF, *STARS Prime Contract - IBM Delivery Order 0002*, April, 1989.

[IBM1280] IBM Systems Integration Division, *Program Administration Plan for STARS*, CDRL Sequence No. 1280, July 31, 1989.

[IBM1540] IBM Systems Integration Division, *Repository Guidebook (Draft)*, CDRL Sequence No. 1540, September 14, 1989.

[IBM460] IBM Systems Integration Division, *Repository Guidelines and Standards*, CDRL Sequence No. 0460, March 17, 1989.

[Boeing330] The Boeing Company, *Repository and Security Plan*, CDRL 0330, December 9, 1988.

[Unisys340] Unisys Corporation, *Reusability Guidelines Draft*, CDRL 0340, February 16, 1989.

[IEEE729] IEEE, *Standard Glossary of Software Engineering Terminology*, ANSI/IEEE STD 729-1983, IEEE Standards Board, September 23, 1983.

[SPCStyle] The Softwre Productivity Consortium, *Ada Quality and Style: Guidelines for Professional Programmers*, June 1989.

[Peters82] Peters, T. J. and R. H. Waterman, Jr., *In Search of Excellence*, Warner Books, 1982.

[Brooks75] Brooks, F. P., Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1975.

Bibliography

The following is a complete bibliography in alphabetic order for this technical report and the attached guides.

STARS Documents

The Boeing Company, *Documentation Requirements for A014 Type CDRLS*, October 1, 1989.

The Boeing Company, *Repository and Security Plan*, CDRL 0330, December 9, 1988.

The Boeing Company, *Repository User's Guide*, March 7, 1989.

The Boeing Company, *Standards and Guidelines for Repository*, CDRL 0320, March 17, 1989.

The Boeing Company, *STARS Repository Acceptance Criteria*, July 21, 1989.

IBM Systems Integration Division, *Consolidated Reusability Guidelines*, CDRL Sequence No. 0380, March 21, 1989.

IBM Systems Integration Division, *Consolidated Technical Development Plan for STARS Competing Prime Contractors*, CDRL Sequence No. 0070, November 11, 1989.

IBM Systems Integration Division, *Draft Policies and Procedures*, CDRL Sequence No. 1460, January 19, 1990.

IBM Systems Integration Division, *DTD Definition: Internal Documentation*, CDRL Sequence No. 0710, January 16, 1989.

IBM Systems Integration Division, *Environment Capability Matrix*, CDRL Sequence No. 0110, March 17, 1989.

IBM Systems Integration Division, *Informal Technical Report on Findings During the Rebuild of Common Capabilities*, CDRL Sequence No. 0340, February 19, 1989.

IBM Systems Integration Division, *Long Term Configuration Management Plan for the STARS Repository*, CDRL Sequence No. 0520, March 17, 1989.

IBM Systems Integration Division, *Practical Aspects of Repository Operations*, CDRL Sequence No. 1440, January 10, 1990.

IBM Systems Integration Division. *Program Administration Plan for STARS*, CDRL Sequence No. 1280, July 31, 1989.

IBM Systems Integration Division, *Quality Assurance/Configuration Management Plan*, CDRL Sequence No. 1320, October 20, 1989.

IBM Systems Integration Division, *Repository Demonstration Informal Report*, CDRL Sequence No. 1610, February 20, 1990.

IBM Systems Integration Division, *Repository Guidebook (Draft)*, CDRL Sequence No. 1540, September 14, 1989.

IBM Systems Integration Division, *Repository Guidelines and Standards*, CDRL Sequence No. 0460, March 17, 1989.

IBM Systems Integration Division, *Repository Operations and Procedures*, CDRL Sequence No. 1470, March 7, 1990.

IBM Systems Integration Division, *Repository Prototype System Specification*, CDRL Sequence No. 1580, February 16, 1990.

IBM Systems Integration Division, *Repository User's Guide*, January 9, 1990.

IBM Systems Integration Division, *Reusability Guidelines*, CDRL Sequence No. 0360, December 17, 1988.

IBM Systems Integration Division, *Reusable Component Data Analysis*, CDRL Sequence No. 0370, February 10, 1989.

IBM Systems Integration Division, *Taxonomy Report*, CDRL Sequence No. 1580, January 19, 1990.

IBM Systems Integration Division, *Version Description Document for the IBM STARS Repository*, CDRL Sequence No. 1600, January 31, 1990.

Naval Resarch Laboratory, *STARS Foundations: Reusability Guidebook*, September, 1986.

Unisys Corporation, *Reusability Guidelines Draft*, CDRL 0340, February 16, 1989.

United Statest Department of Defense, Electronic Systems Division USAF, *STARS Prime Contract - IBM Delivery Order 0002*, April, 1989.

United States Department of Defense, Department of the Air Force, *STARS Competing Primes Lead Contracts Request For Proposal*, F19628-88-R-0011, November 5, 1987.

Other Documents and Articles

Aho, A. V., J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Readding, Mass.: Addison-Wesley, 1974.

Barnes, J.G.P., *Programming in Ada*, 2nd edition. Addison-Wesley Publishers Limited, 1984.

Bentley, J., "Programming Pearls," *Communications of the ACM*, vol. 28, no. 7, July 1985.

Booch, G., *Software Components With Ada.* The Benjamin/Cummings Publishing Company, Inc., 1987.

Brooks, F. P., Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1975.

Burton, B. A., and others, "The Reusable Software Library," *IEEE Software*, July 1987.

Embley, D. W. and S. N. Woodfield, "Cohesion and Coupling for Abstract Data Types," *Proceedings, Sixth Phoenix Conference on Computers and Communications*, Phoenix, Arizona, February, 1987.

EVB Software Engineering, Inc., *Creating Reusable Ada Software*, 1987.

Goguen, J. A., "Reusing and Interconnecting Software Components," *Computer*, February, 1986.

Goodenough, J. and others, *Ada Reusability Guidelines*, SofTech, Inc., April 1985.

IBM, *Common User Access Advanced Interface Design Guide*, SC23-4582-0, June 1989.

IEEE, *Standard Glossary of Software Engineering Terminology*, ANSI/IEEE STD 729-1983, IEEE Standards Board, September 23, 1983.

Kernighan, B. W. and P. J. Plaugher, *The Elements of Programming Style*, Yourdon, Inc., 1978.

LabTek Corp., *Transportability Guideline for Ada Real-Time Software*, U.S. Army HQ CECOM Center for Software Engineering, April 24, 1989.

Matsumoto, Y., "Some Experiences in Promoting Reusable Software Presentation in Higher Abstract Levels," *IEEE Transactions on Software Engineering*, vol. SE-10 (5), September 1984.

Matthews, E. R., *Ada Exception Handling Seminar*, IBM Systems Integration Division, May 1988.

Matthews, E. R., *IBM Federal Systems Division Guide for Reusable Ada Components (Draft)*, September 17, 1987.

Matthews, E. R., "Observations on the Portability of Ada I/O", *ACM SIGAda Ada Letters*, vol. VII, no. 5, September/October 1987.

McDonnell Douglas Astronautics Company, "Overview and Commonality Study Results,", *Common Ada Missile Packages (CAMP)*, AFATL-TR-85-93, May 1986.

McIlroy, M. D., "Mass Produced Software Components," *Report on a conference by the NATO Science Committee*, Garmisch, Germany, October 7-11, 1968.

Mendal, Geoffrey O., "Three Reasons to Avoid the Use Clause," *ACM SIGAda Ada Letters*, vol. VIII, no. 1, January/February 1988.

Merriam-Webster Inc., *Webster's Ninth New Collegiate Dictionary*, Springfield Mass., 1988.

Nielsen, K. W., "Task Coupling and Cohesion in Ada," *ACM SIGAda Ada Letters*, vol. VI, no. 4, July/August 1986.

Nissen, J. and P. Wallis, *Portability and Style in Ada*, Cambridge University Press, 1984.

Pappas, F., *Ada Portability Guidelines*, SofTech, Inc., March 1985.

Peters, T. J. and R. H. Waterman, Jr., *In Search of Excellence*, Warner Books, 1982.

Peterson, A. S., "Coming to Terms with Terminology for Software Reuse," *Reuse in Practice Workshop*, 1989.

Prieto-Diaz, R. and P. Freeman, "Classifying Software for Reusability," *IEEE Software*, January, 1987.

Racine, R., "Why the Use Clause is Beneficial," *ACM SIGAda Ada Letters*, vol. VIII, no. 3, May/June 1988.

U.S. Department of Defense, Ada Joint Program Office, *Rationale for the Design of the Ada Programming Language*, 1984.

U.S. Department of Defense, Ada Joint Program Office, *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A, February, 17 1983.

Rosen, J. P., "In defense of the 'use' clause," *ACM SIGAda Ada Letters*, vol. VII, no. 7, November/December 1987.

Rymer, J. and T. McKeever, *The FSD Ada Style Guide*, 1986.

Software Engineering Institute, "Reuse: Where to Begin and Why," *Affiliates Symposium*, May 2-4, 1989.

SofTech, Inc. *ISEC Portability Guidelines*, December 1985.

SofTech, Inc. *ISEC Reusability Guidelines*, December 1985.

The Software Productivity Consortium, *Ada Quality and Style: Guidelines for Professional Programmers*, June 1989.

Sommerville, I., *Software Engineering*, 3rd. edition, Addison-Wesley, 1989.

St. Dennis, R., P. Stachour, E. Frankowski, and E. Onuegbe, "Measurable Characteristics of Reusable Ada Software," *ACM SIGAda Ada Letters*, vol. VI, no. 2, March/April 1986.

Stevens, W. P., G. J. Myers, and L. L. Constantine, "Structured Design," *IBM Systems Journal*, no. 2., 1974.

Acronyms

The following is a list of acronyms, abbreviations, and similar terms used in this technical report. The attached guides contain separate lists.

Acronym	*Meaning*
Ada	a DoD mandated programming language
AdaMAT	an Ada Metric Analysis Tool by Dynamics Research Corp.
AdaNET	a public domain Ada software reuse service
ASR	Ada Source Repository (also know as SIMTEL-20)
CDRL	Contract Data Requirements List
DID	Data Item Description
DoD	(United States) Department of Defense
IBM	International Business Machines
IEEE	Institute of Electrical and Electronic Engineers
IR40	IBM STARS R-increment task for Repository Integration
NRL	Naval Research Laboratory
Oracle	a commercial relational database product
RAPID	an Army Ada reuse repository system
RFP	Request for Proposal
SAIC	Science Applications International Corporation
SOW	Statement of Work
SPC	Software Productivity Consortium
STARS	Software Technology for Adaptable, Reliable Systems
VAX	a computer system from Digital Equipment Corporation
VDD	Version Description Document
VMS	a proprietary operating system for a VAX

Attachments

Attached are the STARS prime contract R-increment versions of the IBM STARS Repository Guidebook, the STARS Reusability Guidelines, and the IBM STARS Repository User's Guide. We welcome all comments and suggestions that result from your review of this material. Your input will help us achieve our goal of making these guides the primary reference source for users and administrators of the Repository.

Abstract

This technical report (CDRL Sequence No. 1550) addresses subtask IR40.1.1 (Guidelines, Procedures, and Standards) of the STARS Delivery Order Task IR40 (Repository Integration). It includes, as an attachments, the IBM STARS Repository Guidebook, the STARS Reusability Guidelines, and the IBM STARS Repository User's Guide. This report presents the methods used and lessons learned in developing the guides and the Repository. It is a revision of the draft Guidebook (CDRL Sequence No 1540).

The Repository guides define

- How to use the IBM STARS Repository,
- Resources and capabilities of the Repository,
- Component standards for admission to and promotion within the Repository,
- Procedures for submission and usage of repository components,
- Processes involved in managing repository assets, and
- The reuse process in the context of the STARS repository.

Preface

Due to its collective purpose and broad subject coverage, this report and the attached guides contain contributions from numerous individuals and relies on the work presented in many other documents and articles. Key among these are: Wendell Young and Gary Turner, authors of earlier IBM STARS documents; and all the developers in the IBM STARS IR40 team: Matt Bieri, Carol Heymann, Pamela Metcalf, Barbara Morck, Kyle Kennedy, Steve Kutoroff, Mike Puhlmann, and Tom Ward.

This report was originally scheduled to be completed on January 31, 1990, but due to the allocation of personnel to other STARS critical tasks and the extension of the STARS R-increment it was not completed until April, 1990. The attached guides will be updated at the end of the R-increment extension to reflect the final R-increment Repository.

This report was developed by the IBM Systems Integration Division, located at 800 North Frederick Ave., Gaithersburg, MD 20879. Questions or comments should be directed to the author, Robert W. Ekman at (301) 240-6431, or to the IBM STARS Program Office.

Repository Guidebook (Final)
Technical Report

July 2, 1990

Contract No. F19628-88-D-0032
Task IR40: Repository Integration
CDRL Sequence No. 1550

Prepared for:
Electronic Systems Division
Air Force Systems Command, USAF
Hanscomb AFB, MA 01731-5000

Prepared by:

IBM Systems Integration Division
800 North Frederick Ave.
Gaithersburg, MD 20879

Contents

Abstract 1

Preface 2

Introduction 1
Identification 1
Purpose 1
Terminology 1
Task Description 3
Information Sources 3

Guidebook Concept 5
Organization/Structure 5
Content 6
Assumptions 7

Conclusions 8
Lessons Learned 8
Repository Lessons 8
Reusable Component Lessons 10
Software Development Life-Cycle Lessons 12
Recommendations 15

References 16

Bibliography 17

Acronyms 20

Attachments 21

Introduction

Identification

This technical report, IBM Contract Data Requirements List (CDRL) Sequence Number 1550, addresses the Software Technology for Adaptable, Reliable Systems (STARS) Subtask IR40.1: Functionality, as defined in the *STARS Prime Contractor, Delivery Order 2* [DO2], Task IR40: Repository Integration, and satisfies, in part, the *IBM STARS Program Administration Plan* [IBM1280], Subtask IR40.1.1: Guidelines, Procedures, and Standards.

This report is a revision of the *Repository Guidebook (Draft)* "IBM1540", which was delivered under the same subtasks. Refer to the Program Administration Plan for a complete list of deliveries under the IR40 task and its subtasks.

This report follows the format requirements of MIL-STD-847B, as specified in the Technical Report Data Item Description (DID) DI-S-3591/A.

Purpose

This report was prepared to satisfy the delivery requirements of the STARS IR40 Repository Integration task and to provide a place to record the experiences gained from the task. The Guidebook, Guidelines, and User's Guide are attached as a separate stand-alone documents that can be distributed to users of the Repository.

The guides are a collection of descriptions, procedures, and guidelines that covers all views of the IBM STARS Repository, including system access, user roles, repository capabilities, component contributions, component coding, and repository administration. The guidelines and procedures have been applied to the IBM STARS team tasks during the STARS prime contract R-increment and will be applied to IBM STARS tasks in subsequent increments.

Terminology

The following is a list of terms and definitions used in this report. An appreciation of these terms will help in understanding the IR40 Repository Integration task. We have chosen to present them in a descriptive order which supports the relationships between the terms and permits serial reading.

component — A collection of related work products to be used as a consistent set of information. Software work products can include specifications, design, source code, machine code, reports, compilation units, code fragments and other components. A component is the focus of a repository and is intended to be reused. It is the primary type of object found in a repository. Also referred to as an **asset**, or **resource**.

part — An element of a component, such as design documents, source code, test information, and data rights. While a part may be copied or browsed, when stored in a repository it is always associated with a component.

reuse — The application of existing solutions, as captured in components, to a problem other than the one for which they were originally built.

repository	An element of the software engineering environment in which software work products and information about them are stored. Its primary purpose is to support the reuse of components through a process of component submission and extraction. Also referred to as a **reuse library**.
repository system	An instance of the software engineering environment, including computer hardware, operating systems, software tools, standards, and procedures that together provide a complete repository capability. These capabilities include acquiring, storing, managing, retrieving, and dispensing software work products and information about them to potential reusers. A repository system may contain several repositories, each dealing with a different problem domain.
depository	A repository or part of a repository whose contents are deposited as is, with few or no constraints. A depository is primarily a storage facility. Quality and usability are unpredictable; the burden is on the user to find and evaluate useful items.
organized repository	A repository whose contents are documented and organized in a comprehensive manner as components and parts. Finding, reviewing, and extracting components are supported. The quality and usability remain unpredictable.
filtered repository	A repository whose components are subjected to standards on form, content, quality, and consistency. This is not a physical partition from the organized repository, rather it is a documented higher level of confidence in an existing organized component.
certified repository	A repository whose components exhibit the highest level of confidence. A certified component permits the proving of correctness in a solution that reuses the component. Aspects of security, ownership, distribution, and user set take on greater importance.
reuser	A person who reviews the contents of a repository and extracts components for reuse. The common user of a repository. Also referred to as a **user**.
supplier	A person who places components into a repository. Also referred to as a **contributor**, or **submitter**.
librarian	A person who coordinates, organizes, and controls the contents of a repository. Also referred to as a **system administrator, database administrator**, or **repository administrator**. This person frequently provides a first level help for other users.
topic specialist	A person who is responsible for the evaluation of components and their promotion to the filtered or certified status. This person is an expert in the repository domain and has reviewed many of the components in the repository. Also referred to as a **domain expert**, or **technical consultant**. This person frequently provides a second level help for other users, via reference from the librarian.
filtering	The act of evaluating components and promoting them to filtered status.
gatekeeper	An automated capability that facilitates filtering.

These definitions are derived from definitions in several STARS Prime contract documents [IBM380, IBM460, Boeing330, Unisys340], and from several industry publications [IEEE729, SPCStyle]. A complete list of the IBM STARS Repository terminology is given in the Guidebook Glossary.

Using these terms, the IBM STARS Repository is defined as a repository system, that supports the reuse of components in the domain of software engineering environments. The Repository contains a full suite of capabilities and guidelines that are required by users of the system. The Repository supports multiple levels of component confidence (depository, organized, filtered, and certified) and has facilities to promote components through these levels.

Task Description

This report and attached guides are the results of efforts by members of the IBM STARS IR40 task team over the eight month performance period of the task. We conducted numerous reviews, meetings, and discussions on the material presented. We iteratively developed and evaluated operating prototypes to experiment with ideas and validate approaches.

Several existing repositories were reviewed and exercised, including the Ada Source Repository (ASR), the AdaNET information service, the Boeing STARS Repository, and the Army RAPID component repository. These systems provided insight into performance, content, and user issues.

Besides the producing these guides, our activities have

- Refined the state-of-the-art in repository and reuse technology,
- Improved our understanding of operational repository systems,
- Made our experiences available to government and industry through our delivery items, and
- Established an operational repository to support future IBM STARS tasks.

Information Sources

The primary sources for this report and the attached guides were the IBM STARS documents: CDRL 380 *Consolidated Reusability Guidelines* [IBM380], CDRL 460 *Repository Guidelines and Standards* [IBM460], and CDRL 1540 *Repository Guidebook (Draft)* [IBM1540]. These documents were derived from several older documents, some of which were reviewed again during this task. In addition, this report, the guides, and the Repository development were influenced by numerous articles and papers on software reuse and repository technology.

The following is a structured list of the information sources that were used during this task. Complete references for the documents are in the Bibliography for this report. Copies of all the STARS documents can be found in the IBM STARS Repository.

1. STARS Q-Increment Documents
    a. IBM
        1) Task Q12
            a) CDRL 460: *Repository Guidelines*
            b) also, but of lesser importance:
                i. CDRL 470: *Repository System Plan*
                ii. CDRL 480: *Demonstration Script*
                iii. CDRL 490: *Demonstration Results*
                iv. CDRL 500: *Repository Specifications*
                v. CDRL 510: *Repository Implementation Report*
                vi. CDRL 520: *Repository Configuration Control Plan*
                vii. CDRL 530: *Prototype Repository Implementation*
        2) Task Q3
            a) CDRL 110: *Environment Capabilities* (included CDRL 100: *Matrix*, and CDRL 90: *Requirements*)
        3) Task Q9
            a) CDRL 380: *Reusability Report* (included CDRL 370: *Analysis*, and CDRL 360: *Guidelines*)
    b. Boeing
        1) Task Q12
            a) CDRL 320: *Repository Guidelines*
            b) also, but of lesser importance:
                i. CDRL 330: *Initial Capabilities*

ii. CDRL 380: *Configuration Control*
iii. CDRL 400: *Repository Report*
iv. CDRL 420: *Demonstration Plan*

c) peer review of:
i. IBM Task Q3 CDRL 100
ii. IBM Task Q9 CDRL 380

c. Unisys
1) Task Q9
a) CDRL 340: *Reusability Guidelines*

2. STARS R-Increment Documents

a. Task IR40: *Repository Integration Proposal*
b. Task IR40 CDRL 1540: *Repository Guidebook (draft)*
c. Task IR40 CDRL 1580: *Taxonomy Report*
d. Task IR40 CDRL 1590: *Repository Prototype Specifications*
e. Task IR40 CDRL 1600: *Repository Version Description Document*
f. Task IR40 CDRL 1610: *Repository Demonstration*
g. Task IR10 CDRL 1440: *Repository Operations*
h. Task IR10 *Repository User's Guide*
i. Task IR11 CDRL 1460: *Repository Policy and Procedures*
j. Task IR11 CDRL 1470: *Repository Operations and Procedures*

3. Other IBM Documents

a. IBM Systems Integration Division Owego *Reusability Guidelines*
b. IBM Systems Integration Division work in reuse and repositories
c. IBM Corporate work in reuse and repositories

4. Other Key Documents

a. Software Productivity Consortium (SPC) *Ada Style Guidelines*
b. Naval Research Laboratory (NRL) *Reusability Guidelines*
c. SoftTech *Reusability and Portability Guidelines*

5. Tools/Product/System Documentation

a. VAX/VMS
b. AdaMAT
c. Oracle
d. ASR
e. RAPID
f. AdaNET

6. Miscellaneous

a. IBM STARS Program Office notes
b. Developer's notes
c. Feedback from reviewers
d. Operational repository procedures

Guidebook Concept

The goal for the Guidebook is to define the reuse process and gather in one place all the references, standards, guidelines, and procedures related to the operation and use of the IBM STARS Repository. The Guidebook and related guides will be handed out to software engineers who want an introduction to the Repository and will be using the system. The guides will be stored in electronic form on the Repository system.

Organization/Structure

The Guidebook was prepared in separable parts to assure accessibility and convenient use. The arrangement of its constituent parts is a matter of practicality, dictated by the source of the information.

The Guidebook has become a suite of documents, with the Guidebook at the center. This technical report is considered a 'preface' to the Guidebook and not required by any Repository user. The Reusability Guidelines were separated from the Guidebook to provide guidance for component developers. The User's Guide was also made separate from the Guidebook to permit changes in operational aspects of the Repository, without republishing the complete guidebook.

The Guidebook and its related guides are living documents. Each part has a potential for change based on its relationship to the Repository system and reuse as practiced by the IBM STARS team. Updates will be prepared as these changes dictate.

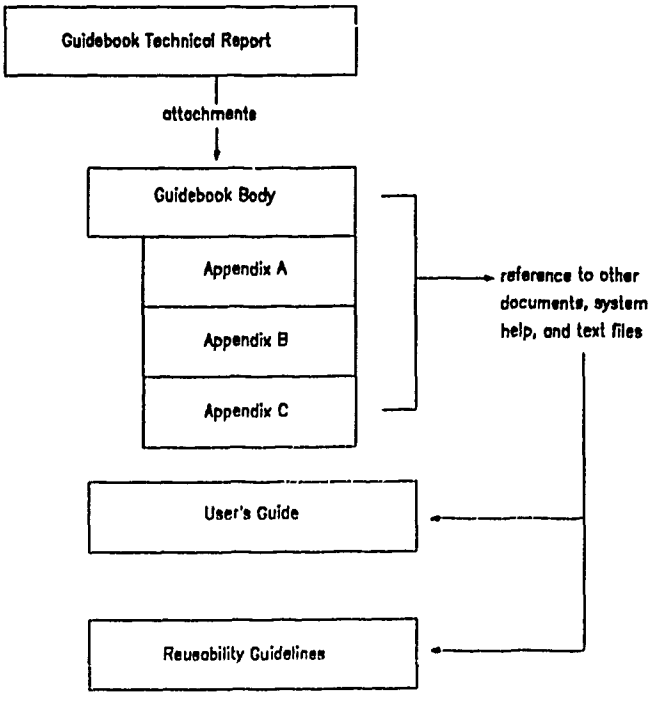

Figure 1. Graphic Model of the Guidebook Parts

In organizing the body of the Guidebook, we considered several dimensions or views of the Repository and the reuse practices. These included:

- the roles of system users (reuser, supplier, administration)
- repository levels (organized, filtered, certified)
- requirements for entry and promotion
- procedures to be followed
- system design and architecture

From discussions of these alternatives, we decided that the *roles* dimension was the best way to present the material, and that roles would be the focus of the body of the Guidebook. The other views tended to have long lists of detail and did not help the user in organizing their approach to reuse. The other views are presented in the appendices or by references to other material.

Content

The Repository Guidebook defines

- Resources and capabilities of the STARS Repository,
- Procedures for submission and usage of repository components,
- Standards for promotion of components within the Repository,
- The processes involved in managing repository assets, including component configuration management, problem tracking, promotion, and entry and exit criteria, and
- The reuse process in the context of the STARS repository, covering the admission, adaptation, integration and testing of reusable components, as well as methods for locating and selecting reusable components.

While producing the Guidebook, reviewers asked for many topics to be included. We realized that attempting to cover all of them would be difficult, but we also felt that most of the request were real needs for some part of the user population. The challenge was to organize the material and structure the documents to give an meaningful impression of the repository, while preserving enough detail to be helpful for a particular user with a specific problem.

The following is the list of topics we addressed in this task and where we decided to place it.

Guidebook material	Location
people roles and activities	body
glossary	body
definitions	body, report
references	body, report
acronyms	body, report
relationship to STARS tasks	report
guidebook concept	report
assumptions	report
lessons learned	report
bibliography	report
summary of resources and services	appendix
gate definitions (filter list)	appendix
user directions	appendix, attachment
coding guidelines for reusability	appendix, attachment
operational policies of our system	appendix, reference
data requirements and database tables	reference
help information	reference
system screen images	reference
how to do reuse	reference
where reuse fits in lifecycle	reference
configuration management	reference
access security	reference

Assumptions

The following assumptions pertain to the guidelines and procedures presented in the Guidebook.

- The Guidebook and associated guides support reuse within the context of a software intensive project. They may be used to establish a starter set of guidelines, but does not form a complete set. Your project should modify and augment these guidelines to address domain and system specific issues.
- Issues of security, proprietary rights, and component ownership are beyond the scope of this version of the Guidebook, but may be found in referenced documents.
- The coding guidelines are directed primarily toward Ada software engineering. While some guidance may apply to a broader class of components, the focus is on Ada software. The guidelines can thus treat specific aspects of Ada that relate to reusability.
- No assumptions are made about design methodology and relationships to reuse.
- Reuse is assumed to include tailoring of reusable components. It is likely that a significant amount of reuse will involve some tailoring of the component code. Thus, there are guidelines to promote tailorability. Note that *tailoring* should be distinguished from the broader term *maintenance*, which not only includes tailoring but also includes the idea of corrective maintenance (fixing bugs).

Conclusions

The IBM STARS IR40 task team has worked for nine months developing the prototype repository system that is documented by the Guidebook. The team has included up to ten engineers, in three different locations, developing the system on several different platforms. The team developed four releases of the system following an iterative refinement prototyping method. The experience gained during this task is the basis for the conclusions in this report.

Lessons Learned

Several situations arose during the writing of the Guidebook and the development of the Repository which we believe contain "lessons learned" that could be applied by the greater software development community. We have collected these lessons and classified them according to software engineering technology areas. In many cases, the lessons were previously stated in draft documents and status reports. Our purpose in this report is to gather all the IR40 team lessons learned into one place.

The lessons are stated as a recognized problem or situation, followed by our observations on the problem, an implemented solution, or alternative potential solutions.

Repository Lessons

At a high level, developing a repository has issues and design similar to the development of any on-line application with a large database. But when you consider a repository system as a special element in a software engineering environment, unique problems appear.

From an object-oriented view, the life-cycle of objects in a repository is similar to the life-cycle of objects in a software engineering configuration management system. The differences are that the repository objects are less frequently updated, that there are weaker relationships between the objects, and that ownership is harder to establish. A software engineering environment deals with objects as a project library, while a repository deals with objects as a public library.

- *It is hard to describe the Repository due to the varying experiences of the users and the different roles they play in the reuse paradigm.*

  The critical information in a repository can be viewed from many different perspectives. For example, the coding guidelines can be viewed from the perspective of the developer of reusable components, the submitter who presents the component for admission to the Repository, the topic specialist who analyzes the component, and the reuser who extracts the component from the Repository. The developer follows the guidelines in creating the initial work product. The submitter and topic specialist reference the coding guidelines to confirm that they have been adequately followed. The reuser will want to know what coding guidelines are preferred and supported by the repository to facilitate the usability analysis.

  To address this situation, we organized the body of the Guidebook based on user roles. We also organized the Repository capabilities based on user roles. This organization is apparent in the selection menus and the system design.

- *There were few aids to make it easy to follow guidelines and there were no aids that support different user roles.*

  Each role is defined with a set of procedures which generally refer to one or more sets of guidelines. The guidelines are often too extensive or too broad for regular reference. We found that checklists can be used to promote consistency and completeness and to customize guidelines and procedures to a particular environment or application. The checklists made it possible to provide a single set of general

guidelines and use the more succinct checklist to place emphasis on the important guidance. For example, the coding guidelines are customized for the component developer by the coding guidelines checklist. Further customizing could be achieved by providing portability and reusability checklists which emphasize those coding guidelines that topic specialists should consider in evaluating a component for admission to the repository.

- *The relationships between Guidebook information, its sources, and users are complex.*

  The intellectual, physical, and temporal relationships among the Guidebook information, its sources, and its users are quite complex, making it difficult to provide the information in a meaningful and manageable manner. It seems clear at this point, that the CDRL delivery mechanism is not the complete answer, since the information in deliveries, as a whole, changes continuously. The fundamental problem is the paradox that specific information is both more useful to Repository users and more volatile than more general information. For example, the names of individuals identified as points of contact or assigned to other roles are more likely to change than the role abstraction.

  The potential users of the information contained in this Guidebook are many and diverse. For this information to be used, it is necessary to achieve several goals. It must be provided accurately, when it is needed, where it is needed, in a form that is meaningful to the user of the information, and in a manner that is as unobtrusive as possible. Indications (such as time-stamp or version number) should also be provided to give the user confidence that the information is as timely and accurate as it needs to be. This implies that information updates must be rapidly disseminated to end users. Ways must be found to improve the packaging and delivery of the Guidebook and its information content.

- *It was difficult to synchronize updates to information, even when a direct relationship was understood.*

  Information sources also influence the packaging objectives for the information since it is desirable (though not necessary) for information provided by a given source to be (logically) co-located and for updates to occur as atomic operations to ensure the continuing validity of the complete document.

  This becomes increasingly important as various manual functions are automated. We have used information modelling techniques to develop our understanding of the relationships and predict the synchronization problems.

- *The development guidelines, on-line definition of the guidelines, and the automation of component review regarding the guidelines were difficult to coordinate.*

  In establishing a complete repository, we developed guidelines for components and procedures for the component life-cycle within the Repository. These guidelines and procedures appeared in several places, such as the Guidebook, on-line help, and rules for Ada code metric analysis. In addition, they were refined and modified over time.

  This situation made it difficult to coordinate the guidelines development. At times the guidance in one place contradicted the guidance in another place. By the end of the R-increment, we believe most of the irregularities have been addressed, but we have not established an easy way to keep the guidance coordinated and get a reasonable level of consensus.

- *An undefined glossary made documentation, discussions, and component evaluation difficult.*

  In the Repository, we needed to distinguish terms from their more general English definitions. We constructed a structure of terms and classifications within those terms, but we had difficulty in communicating our meanings.

  We found many competing definitions for the same words. Similar type work in reuse and repositories is being conducted by many organizations and they have published their glossaries. These definitions overloaded our definitions and caused misunderstanding. Further, we contributed to the confusion due to our own refinement and elaboration of our definitions over time. These problems are not unusual in leading-edge efforts, but they are areas for concern and require attention.

  Our answer to the terminology problems was to include our definitions in several key documents and reports. We reviewed some of the competing glossary lists and adopted their definitions where

appropriate. But in the end, we were left with a set of definitions that work for us, but are neither "fish nor fowl" in any official world.

A similar issue appeared when reviewing documentation for software components. For example, it was difficult to understand and evaluate the "Dates" software package without a good understanding of horological terms, such as Julian date.

To answer the component understanding problem, we have suggested that components contain a glossary of keywords in all related documentation. There is a direct relationship between the documentation of the component glossary and the readability of the declarative section of the code.

- *The linear nature of hard-copy documents does not translate to on-line review very easily.*

  Books, manuals, and other hard-copy documents are inherently linear in physical organization. This results in difficult choices: Is it better to repeat textual information at the risk of annoying the reader who encounters repeated text, or is it better to provide references among the singular instance of a unit of text much like is done in an encyclopedia, putting the burden of locating the information on the reader?

  In the Guidebook, we chose the reference approach. This choice also reduced the maintenance and new material problems, much like subunits in a programming language.

  Some alternatives to the linear organization are network links between material as implemented in hypertext systems, and hierarchy tree models as implemented in many on-line help systems. These alternatives are desirable organizations for the Repository information system, but were displaced by more important tasks.

- *Automated component evaluation was only partially successful.*

  We attempted to construct an automatic filter for components going into the Repository and for evaluation of components within the the Repository. We found it time consuming and difficult to construct these automatic filters, and we found that they ultimately would only do part of the job.

  To evaluate the compilation status of a component, the bulk of the effort was consumed by review of the compilation order information, location of external packages and libraries, and similar type tasks. Running the compiler was trivial, and automation of just the compilation step did not save much effort.

  The review of documentation was all manual. We were not able to locate or build any tool to help with documentation evaluation. Because of the variety of documentation standards and styles, it is unlikely this area will be automated soon. There are some tasks, such as grammar and spelling checks, that can be supported by tools, but they do not help with the evaluation of documentation completeness.

  Review of Ada code was partially automated with the help of the tool AdaMAT, by Dynamics Research Corporation (DRC). This tool gave us counts of constructs that are considered "good" against a count of where the construct could have been used. The AdaMAT reports helped to suggest places and issues to review, but could not substitute for human inspection of the code.

Reusable Component Lessons

We populated the IBM STARS Repository with hundreds of components and thousands of related parts. This population task was usually accomplished by system administrative engineers, and occasionally by the component authors.

Some of the following lessons may seem obvious and the answers may appear to be the application of rigor and management techniques. But this is the area where we had the least success. We originally planned to have many more components and parts in the final repository. Every time we sat down to really build up the repository content, other things got in the way. The database structure kept changing, the submit mechanism was too hard to use, or the task was just too boring and labor intensive.

The following lessons present major issues from our evaluation of the problems in repository content management.

- *It is difficult to successfully and completely contribute a component from a third party.*

  The process of organizing the components in a repository is more difficult when the components are extracted from a source, rather than submitted by a supplier. For example, some of the components from the Ada Source Repository (ASR) have been organized on the IBM STARS repository. These components took much longer to organize than the components submitted by the STARS prime contractors, because the ASR librarian was not involved in meeting the IBM STARS repository entry requirements. On the other hand, some of the STARS prime components were well known by the people doing the entry to repository, and the time required to organize them was significantly less.

  This observation led us to put a greater emphasis on having the component author do the submission to the repository. The person who knows the most about a component is its author, and the repository submit process should take advantage of this knowledge. The submit process must be easy to use, capturing a large amount of information, which would be harder to acquire later on. A clear but not extensive set of guidelines and procedures must be established, and they must be supported by the system capabilities, especially at the component entry time.

  This observation also leads us to question the value of third party repositories that are currently available, such as the ASR, AdaNET, and the Ada Information Clearinghouse. These systems freely pass along components and information, and they support general communications, which is much needed in the Ada technology community. But without the explicit involvement of the authors in the cataloging and depositing of the components, the confidence is lowered in the ownership, completeness, and currency of the component. And confidence in the extracted component is significant in the success of reuse.

- *There is little incentive for engineers to contribute their work products to a repository outside their current project.*

  This lesson is not new. We have just recognized it again, and we want to reiterate the problem. A good reuse repository must contain some incentives that make people want to supply components for reuse. Further, our observations revealed that components that do not have the force of an incentivized supplier behind them, are probably not good candidates for a reuse repository.

  While not being terribly successful, we have attempted to place a repository submission subtask in all major tasks. The concept is that one of the measurements in completing a task is the submission of reusable components out of the task's workproducts into the repository. This is the model that we will follow in future IBM STARS tasks.

  A major disincentive to engineers in our repository model is the effort required to contribute a component. It takes as much as four hours to contribute a moderate size component. This seems excessive, but we believe it is justified. The effort to supply components is inversely proportional to the effort to reuse, and a primary objective of a repository system is to reduce the effort to reuse. A balance is required between the two efforts. A factor in this balance is the realization that supply is a one-time effort and reuse is a recurring effort. Each reuse increases you return on supply cost investment.

- *The names of components and the names within components confused the repository administration and reusers.*

  Contributors frequently used names that are only meaningful to their environments. The names may be imbedded in their contributions and don't "flow through" to the repository system, or to the reuser's development system.

  This problem has became more apparent during final reviews of the Repository code and the Repository content. It can be a serious inhibitor to software reuse.

  We have established name guidance for the Repository development to reduce this problem. The initial guidelines are in the Guidebook. We anticipate refinement of these guidelines and additional guidelines as we learn more about the problem.

- *Software components without compilation order information are almost useless for the reuser.*

  This lesson may seem obvious, but some contributions to our repository (and many contributions to other repositories) have not included compilation order information. The submitter assumes the component is simple enough, such that anyone could compile it, or they rely on the naming convention to indicate compilation order. In some cases, it may be acceptable, but the absence of the compilation order information almost always causes confusion.

  Our guidelines require compilation order information and it is part of the evaluation process followed by the topic specialists. We do not bar contributions without compilation order information, but the components without it will unlikely make it to the filtered state.

- *It is difficult to store assets independent of a repository data base.*

  During the extension to the R increment, an attempt was made to store components in tagged SGML files, and then use a Load tool to store the components in the STARS data base. The purpose was to develop a repository-independent format for components. The information model for a specific release of the repository could change, but the old assets in SGML format could be reloaded by a modification to the load tool. The concept of saving assets in an SGML format also was thought to have use for exchanging assets between repositories.

  The flaw in the idea was that assets referenced other assets, and assets referenced other information in the repository, such as facet term, organization, and owner name. In order for the asset to be independent of a repository, all this information has to be carried along with the asset. This information is very voluminous, and would cause asset load time to be excessively long.

Software Development Life-Cycle Lessons

During the development of the IR40 prototype repository system, we followed a rapid iterative refinement methodology. We did not follow the conventional waterfall life-cycle, rather we followed the emerging software-first life cycle. We eliminated almost all "standard" documents and took many shortcuts in order to produce the greatest amount of operating prototype capability in the shortest time. We wanted to produce the "look and feel" of a repository, based on a general environment architecture, with standard interfaces, all running on real data in a significantly populated database.

We believe the process we followed was acceptable and produced a cost effective repository system. We plan to continue to experiment with this process in future IBM STARS tasks.

We anticipated most of the problems we encountered, since we recognized our deviation from the norm, but we were surprised at how much effort and time was required to address them. There was always a strong feeling to stay with whatever we had, in both software and methods. There was a continuous and uneasy balance between resources available and planned tasks.

- *Operational, development, and test environments interfered with each other.*

  The Repository not only was developed in several locations and on several platforms, but also contained several parts which did not come together until we build an operational system, such as the database, menu system, and search mechanism. This is not unique to this project, but it did cause stress on the development cycle.

  We had to develop multiple databases, and populate them several times. Separate test environments needed to be maintained to allow testing to proceed without disruption of the existing repository data base.

  We developed several layers of the architecture simultaneously. To accomplish this, we copied significant amounts of source code into several development environments. Each environment had naming restrictions which forced us to use different naming patterns for the same source elements. These duplicate systems created a versioning maintenance problem. This also lowered our confidence with the final product, because we could not guarantee that the migrated elements were unmodified.

- *Too few people tried the system too late for changes.*

  The development model we followed called for frequent execution of the evolving system by a wide user population. We accomplished this through full system releases to the STARS community and partial prototypes for a narrow set of users. We either demonstrated the prototype or asked the user to experiment with it by themselves. We captured remarks from demonstrations and problem reports from individual users, as our feedback mechanism.

  The problem we had was in striking a balance between preparing clean prototypes for a large set of users and rapid prototypes for a controlled group. Even though we maintained operational parts of the system throughout the task period, it still took significant efforts to put together fully operational systems. Complete systems were needed for the user to experience and appreciate the set of capabilities. In some cases, we put together partial systems which demonstrated particular ideas. These prototypes proved useful, but only to a small set of intimately involved users.

  We did not uncover any simple way to approach this balancing effort. Just continual attention to the basics of rapid prototyping is all we can offer as a lesson. Rapid iterative development is more successful if early prototype capabilities can be made available to a wide set of experienced users who have access to immediate and easy to use feedback mechanisms.

- *The Repository user interfaces were not consistent among embedded tools.*

  Rapid prototyping and integration frequently leads to systems with "bumpy" user interfaces. In constructing a rapid prototype, you must reuse components that were written for widely different situations. The user interfaces that accompany these components are usually inconsistent. They were written to take advantage of the perceived users experience and operational system. If, when searching for reusable components, you stay only within a single domain and platform, you will reduce this problem, but you will not be able to take advantage of other reuseable components. The development cycle becomes a trade-off between capability, usability, and time.

  During the development of the Repository, we concentrated on the balance between capability and usability. We attempted to extend the system capability with less regard for usability. This approach was helpful in developing the overall architecture and design, but when it came time to demonstrate the system, we discovered usability problems. The problems with usability caused us to lose "user hours" and reduced the feedback on the prototype, and thus reducing the benefit of this life-cycle.

- *Parts of the Repository system did not have clear ownership.*

  The final operational Repository is not a single unit, but rather a collection of executables, held together by a menu selection mechanism, relying on a database for much of the display window content. Included in the Repository are the Guidebook, guidelines, and numerous help screens. A significant amount of material was reused from the STARS Q-increment and incorporated into the Repository system. Furthermore, over the repository development period, there was considerable turnover in the IR40 team. These facts lead to situations in which there was uncertain ownership of elements of the Repository system.

  While the ownership problems were cleared up by the end of the R-increment, it was a noteworthy issue during development. Without clear ownership, rapid development can get out-of-hand. Effective configuration management methods should be used to get the most out of iterative development. When you reuse material that does not have clear ownership, you must be willing to take on ownership. You need to be concerned with ownership of guidelines and procedures to the same level as ownership of software.

- *Quality assurance was not consistent or occurred too late.*

  Even in a rapid prototype development cycle, as performed by the IR40 team, an independent quality assurance audit needs to be conducted. The audits conducted on the Repository system, brought forth several potential problems that were taken care of before reaching the users. But the development cycle used on the Repository did affect the quality assurance tasks.

The timeliness of the quality assurance tasks was a problem. Frequently, we were well into the next iteration before the assurance tasks were complete. The system had matured and obviated many of the reported problems.

The second problem we had was consistency across the multiple sites and development platforms. Each group had their own way of handling quality assurance. The pressure to develop and demonstrate quickly, as called for in a rapid development cycle, dictated that we skip a more complete and orderly approach.

- *User initiated demonstrations were not effective.*

Two methods were used to get reviews of the Repository prototypes: general broadcasts to the user population that a feature or release was available, and personal face-to-face demonstrations by the developers. The mechanism of feedback for the general release was a problem reporting system and electronic mail notes back to the developers. The mechanism of feedback for the personal demonstrations were comments during the presentations and observations of the users actions. While feedback from both methods was helpful, the personal demonstrations had faster feedback and gave a better sense of the value of the feature being demonstrated.

Users seldom took time out of their busy schedule to exercise a prototype or new release under their own initiative. Unless a capability was absolutely needed to complete their tasks, we received very little response from the user community. Even when the effort was planned as peer review in their primary task, the self initiated exercise of a prototype was too late to affect development. The problem reporting and electronic mail feedback did provide a documented trail of problems and a measurable item. But the nature of the reports and notes were too detailed for prototype evaluation, and focused on errors and screen content. Serious problems and alternative solutions generally did not appear in this method of feedback.

Face-to-face demonstrations were conducted in offices between developers and users, in meetings with management and architects, in classroom environments, and at conferences to a wide audience. Very little specific documented feedback resulted from these demonstrations, but user impressions were important in evaluating the success of the prototype.

- *Communications were difficult in the IR40 development team.*

Part of the design and plan of the IR40 task was to experiment with distributed development among locations, development environments, and system architecture. But because of this distributed organization and structure, inadequate communications between the team members was a problem. Tom Peters [Peters82] and Fred Brooks [Brooks75] see communications as the most critical element in the success and excellence of a project. Our experience in the IR40 task agrees with their assertions.

We were not able to eliminate communications problems, but we did address the problem. Our approaches to reduce communication problems were to

- Use travel to the degree permitted in the contract,
- Take advantage of other travel plans to coordinate our work,
- Use conference calls, and
- Use electronic mail, especially over phone lines via the repository system.

A key element in a software development effort is the configuration management system. It is a common notion that these systems help with project communications. Unfortunately, because of the heterogeneous nature of the IBM STARS team network, configuration management systems actually hurt sharing information and communication among the team members. Each group within the team used locally available configuration management methods. The manual import/export capabilities of these systems made it difficult for team members outside the local group to review development.

Recommendations

In summary, the "lessons learned" can be stated in positive action statements as follows:

1. Repository Lessons
    a. Describe the Repository from user views.
    b. Establish guidance using support tools based on user views.
    c. Define relationships between user guidance information.
    d. Synchronize updates to user information.
    e. Coordinate guidance with automated tools.
    f. Publicize a glossary of Repository terms; supply a glossary with components.
    g. Use reference pointers to information sources.
    h. Plan for manual component evaluation, even with automated tools.
2. Reusable Component Lessons
    a. Get the component author or owner involved in the submit process.
    b. Seek an incentive for engineers to contribute their work products.
    c. Pay special attention to component names and embedded names.
    d. Supply compilation order information with components.
3. Software Development Life-Cycle Lessons
    a. Clearly separate operational, development, and test environments.
    b. Involved a large number of reviewers early in the development cycle.
    c. Strive for consistent user interfaces.
    d. Establish clear ownership of developed workproducts.
    e. Apply quality assurance consistently, even in rapid iterative development.
    f. Use controlled small group demonstrations.
    g. Spend significant effort on communications within the development team.

In addition, the development of the Repository and associated Guidebook has lead to the following general recommendations.

- Spend more time with the users, or your user interface will be uninviting.
- Spend effort smoothing the contribution mechanism or you will not have anything to reuse.
- Put aside significant time and funds to populate the repository and evaluate components. Our experience suggests about:
    - 1hr per simple component (component with no dependency on other components) for submission to the repository.
    - 4hrs per simple component for evaluation and filtering.
    - 20hrs per simple component for certification.

References

The following is a list of references in this technical report in the order that they are cited. The attached guides contain separate lists.

[IBM370] IBM Systems Integration Division, *Reusable Component Data Analysis*, CDRL Sequence No. 0370, February 10, 1989.

[IBM380] IBM Systems Integration Division, *Consolidated Reusability Guidelines*, CDRL Sequence No. 0380, March 21, 1989.

[DO2] United States Department of Defense, Electronic Systems Division USAF, *STARS Prime Contract - IBM Delivery Order 0002*, April, 1989.

[IBM1280] IBM Systems Integration Division, *Program Administration Plan for STARS*, CDRL Sequence No. 1280, July 31, 1989.

[IBM1540] IBM Systems Integration Division, *Repository Guidebook (Draft)*, CDRL Sequence No. 1540, September 14, 1989.

[IBM460] IBM Systems Integration Division, *Repository Guidelines and Standards*, CDRL Sequence No. 0460, March 17, 1989.

[Boeing330] The Boeing Company, *Repository and Security Plan*, CDRL 0330, December 9, 1988.

[Unisys340] Unisys Corporation, *Reusability Guidelines Draft*, CDRL 0340, February 16, 1989.

[IEEE729] IEEE, *Standard Glossary of Software Engineering Terminology*, ANSI/IEEE STD 729-1983, IEEE Standards Board, September 23, 1983.

[SPCStyle] The Softwre Productivity Consortium, *Ada Quality and Style: Guidelines for Professional Programmers*, June 1989.

[Peters82] Peters, T. J. and R. H. Waterman, Jr., *In Search of Excellence*, Warner Books, 1982.

[Brooks75] Brooks, F. P., Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1975.

Bibliography

The following is a complete bibliography in alphabetic order for this technical report and the attached guides.

STARS Documents

The Boeing Company, *Documentation Requirements for A014 Type CDRLS*, October 1, 1989.

The Boeing Company, *Repository and Security Plan*, CDRL 0330, December 9, 1988.

The Boeing Company, *Repository User's Guide*, March 7, 1989.

The Boeing Company, *Standards and Guidelines for Repository*, CDRL 0320, March 17, 1989.

The Boeing Company, *STARS Repository Acceptance Criteria*, July 21, 1989.

IBM Systems Integration Division, *Consolidated Reusability Guidelines*, CDRL Sequence No. 0380, March 21, 1989.

IBM Systems Integration Division, *Consolidated Technical Development Plan for STARS Competing Prime Contractors*, CDRL Sequence No. 0070, November 11, 1989.

IBM Systems Integration Division, *Draft Policies and Procedures*, CDRL Sequence No. 1460, January 19, 1990.

IBM Systems Integration Division, *DTD Definition: Internal Documentation*, CDRL Sequence No. 0710, January 16, 1989.

IBM Systems Integration Division, *Environment Capability Matrix*, CDRL Sequence No. 0110, March 17, 1989.

IBM Systems Integration Division, *Informal Technical Report on Findings During the Rebuild of Common Capabilities*, CDRL Sequence No. 0340, February 19, 1989.

IBM Systems Integration Division, *Long Term Configuration Management Plan for the STARS Repository*, CDRL Sequence No. 0520, March 17, 1989.

IBM Systems Integration Division, *Practical Aspects of Repository Operations*, CDRL Sequence No. 1440, January 10, 1990.

IBM Systems Integration Division, *Program Administration Plan for STARS*, CDRL Sequence No. 1280, July 31, 1989.

IBM Systems Integration Division, *Quality Assurance/Configuration Management Plan*, CDRL Sequence No. 1320, October 20, 1989.

IBM Systems Integration Division, *Repository Demonstration Informal Report*, CDRL Sequence No. 1610, February 20, 1990.

IBM Systems Integration Division, *Repository Guidebook (Draft)*, CDRL Sequence No. 1540, September 14, 1989.

IBM Systems Integration Division, *Repository Guidelines and Standards*, CDRL Sequence No. 0460, March 17, 1989.

IBM Systems Integration Division, *Repository Operations and Procedures*, CDRL Sequence No. 1470, March 7, 1990.

IBM Systems Integration Division, *Repository Prototype System Specification*, CDRL Sequence No. 1580, February 16, 1990.

IBM Systems Integration Division, *Repository User's Guide*, January 9, 1990.

IBM Systems Integration Division, *Reusability Guidelines*, CDRL Sequence No. 0360, December 17, 1988.

IBM Systems Integration Division, *Reusable Component Data Analysis*, CDRL Sequence No. 0370, February 10, 1989.

IBM Systems Integration Division, *Taxonomy Report*, CDRL Sequence No. 1580, January 19, 1990.

IBM Systems Integration Division, *Version Description Document for the IBM STARS Repository*, CDRL Sequence No. 1600, January 31, 1990.

Naval Resarch Laboratory, *STARS Foundations: Reusability Guidebook*, September, 1986.

Unisys Corporation, *Reusability Guidelines Draft*, CDRL 0340, February 16, 1989.

United Statest Department of Defense, Electronic Systems Division USAF, *STARS Prime Contract - IBM Delivery Order 0002*, April, 1989.

United States Department of Defense, Department of the Air Force, *STARS Competing Primes Lead Contracts Request For Proposal*, F19628-88-R-0011, November 5, 1987.

Other Documents and Articles

Aho, A. V., J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Readding, Mass.: Addison-Wesley, 1974.

Barnes, J.G.P., *Programming in Ada*, 2nd edition. Addison-Wesley Publishers Limited, 1984.

Bentley, J., "Programming Pearls," *Communications of the ACM*, vol. 28, no. 7, July 1985.

Booch, G., *Software Components With Ada*. The Benjamin/Cummings Publishing Company, Inc., 1987.

Brooks, F. P., Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1975.

Burton, B. A., and others, "The Reusable Software Library," *IEEE Software*, July 1987.

Embley, D. W. and S. N. Woodfield, "Cohesion and Coupling for Abstract Data Types," *Proceedings, Sixth Phoenix Conference on Computers and Communications*, Phoenix, Arizona, February, 1987.

EVB Software Engineering, Inc., *Creating Reusable Ada Software*, 1987.

Goguen, J. A., "Reusing and Interconnecting Software Components," *Computer*, February, 1986.

Goodenough, J. and others, *Ada Reusability Guidelines*, SofTech, Inc., April 1985.

IBM, *Common User Access Advanced Interface Design Guide*, SC23-4582-0, June 1989.

IEEE, *Standard Glossary of Software Engineering Terminology*, ANSI/IEEE STD 729-1983, IEEE Standards Board, September 23, 1983.

Kernighan, B. W. and P. J. Plaugher, *The Elements of Programming Style*, Yourdon, Inc., 1978.

LabTek Corp., *Transportability Guideline for Ada Real-Time Software*, U.S. Army HQ CECOM Center for Software Engineering, April 24, 1989.

Matsumoto, Y., "Some Experiences in Promoting Reusable Software Presentation in Higher Abstract Levels," *IEEE Transactions on Software Engineering*, vol. SE-10 (5). September 1984.

Matthews, E. R., *Ada Exception Handling Seminar*, IBM Systems Integration Division, May 1988.

Matthews, E. R., *IBM Federal Systems Division Guide for Reusable Ada Components (Draft)*, September 17, 1987.

Matthews, E. R., "Observations on the Portability of Ada I/O," *ACM SIGAda Ada Letters*, vol. VII, no. 5, September/October 1987.

McDonnell Douglas Astronautics Company, "Overview and Commonality Study Results,," *Common Ada Missile Packages (CAMP)*, AFATL-TR-85-93, May 1986.

McIlroy, M. D., "Mass Produced Software Components," *Report on a conference by the NATO Science Committee*, Garmisch, Germany, October 7-11, 1968.

Mendal, Geoffrey O., "Three Reasons to Avoid the Use Clause," *ACM SIGAda Ada Letters*, vol. VIII, no. 1, January/February 1988.

Merriam-Webster Inc., *Webster's Ninth New Collegiate Dictionary*, Springfield Mass., 1988.

Nielsen, K. W., "Task Coupling and Cohesion in Ada," *ACM SIGAda Ada Letters*, vol. VI, no. 4, July/August 1986.

Nissen, J. and P. Wallis, *Portability and Style in Ada*, Cambridge University Press, 1984.

Pappas, F., *Ada Portability Guidelines*, SofTech, Inc., March 1985.

Peters, T. J. and R. H. Waterman, Jr., *In Search of Excellence*, Warner Books, 1982.

Peterson, A. S., "Coming to Terms with Terminology for Software Reuse," *Reuse in Practice Workshop*, 1989.

Prieto-Diaz, R. and P. Freeman, "Classifying Software for Reusability," *IEEE Software*, January, 1987.

Racine, R., "Why the Use Clause is Beneficial," *ACM SIGAda Ada Letters*, vol. VIII, no. 3, May/June 1988.

U.S. Department of Defense, Ada Joint Program Office, *Rationale for the Design of the Ada Programming Language*, 1984.

U.S. Department of Defense, Ada Joint Program Office, *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A, February, 17 1983.

Rosen, J. P., "In defense of the 'use' clause," *ACM SIGAda Ada Letters*, vol. VII, no. 7, November/December 1987.

Rymer, J. and T. McKeever, *The FSD Ada Style Guide*, 1986.

Software Engineering Institute, "Reuse: Where to Begin and Why," *Affiliates Symposium*, May 2-4, 1989.

SofTech, Inc. *ISEC Portability Guidelines*, December 1985.

SofTech, Inc. *ISEC Reusability Guidelines*, December 1985.

The Software Productivity Consortium, *Ada Quality and Style: Guidelines for Professional Programmers*, June 1989.

Sommerville, I., *Software Engineering*, 3rd. edition, Addison-Wesley, 1989.

St. Dennis, R., P. Stachour, E. Frankowski, and E. Onuegbe, "Measurable Characteristics of Reusable Ada Software," *ACM SIGAda Ada Letters*, vol. VI, no. 2, March/April 1986.

Stevens, W. P., G. J. Myers, and L. L. Constantine, "Structured Design," *IBM Systems Journal*, no. 2., 1974.

Acronyms

The following is a list of acronyms, abbreviations, and similar terms used in this technical report. The attached guides contain separate lists.

Acronym	Meaning
Ada	a DoD mandated programming language
AdaMAT	an Ada Metric Analysis Tool by Dynamics Research Corp.
AdaNET	a public domain Ada software reuse service
ASR	Ada Source Repository (also know as SIMTEL-20)
CDRL	Contract Data Requirements List
DID	Data Item Description
DoD	(United States) Department of Defense
IBM	International Business Machines
IEEE	Institute of Electrical and Electronic Engineers
IR40	IBM STARS R-increment task for Repository Integration
NRL	Naval Research Laboratory
Oracle	a commercial relational database product
RAPID	an Army Ada reuse repository system
RFP	Request for Proposal
SAIC	Science Applications International Corporation
SOW	Statement of Work
SPC	Software Productivity Consortium
STARS	Software Technology for Adaptable, Reliable Systems
VAX	a computer system from Digital Equipment Corporation
VDD	Version Description Document
VMS	a proprietary operating system for a VAX

Attachments

Attached are the STARS prime contract R-increment versions of the IBM STARS Repository Guidebook, the STARS Reusability Guidelines, and the IBM STARS Repository User's Guide. We welcome all comments and suggestions that result from your review of this material. Your input will help us achieve our goal of making these guides the primary reference source for users and administrators of the Repository.